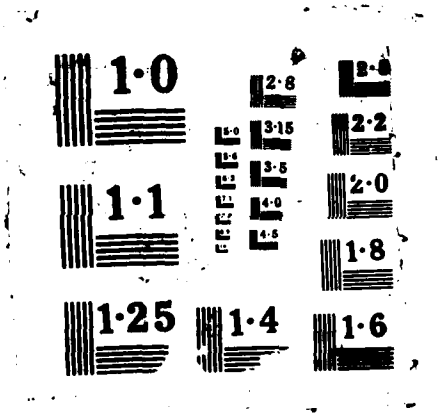
AD-A185 172 NEAR-HORN PROLOG(U) DUKE UNIV DURHAM NC DEPT OF COMPUTER SCIENCE D W LOVELAND 05 MAY 87 CS-1987-14 ARO-21213.3-MA DAAG29-84-K-0072 1/1

UNCLASSIFIED F/G 12/5 NL


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·8
3·15
3·5
4·0
4·5

ARO 21213.3-MA

CS-1987-14

NEAR-HORN PROLOG

D.W. Loveland

Department of Computer Science
Duke University

May 5, 1987

DEPARTMENT
OF
COMPUTER SCIENCE

DUKE UNIVERSITY

2

CS-1987-14

NEAR-HORN PROLOG

D.W. Loveland

Department of Computer Science
Duke University

May 5, 1987

NEAR-HORN PROLOG

D.W. Loveland
Computer Science Department
Duke University
Durham, NC 27706 USA

Abstract. We propose an extension to Prolog that handles clause sets that are Horn and almost-Horn in a manner that emphasizes clarity, efficiency, and minimal deviation from standard Prolog. Although one version of near-Horn Prolog is complete under appropriate search strategies, we focus on incomplete variants that allow fast processing. Negation in the program is handled correctly, although within a positive implication logic extension of Horn logic. Processing speed degradation is directly proportional to the distance the program is from a Horn clause set.

This research has been partially supported by the Army Research Office under Grant DAAG29-84-K-0072.

This paper will be presented at the *Fourth International Conference on Logic Programming*, May, 1987, in Melbourne, Australia.

1. INTRODUCTION.

An extension to standard Prolog is presented that handles Horn clause sets in the usual manner of Prolog and handles a further class of problems that permits disjunctive conclusion (multiple heads) and negated facts. The system we will emphasize is a very natural extension of Prolog and promises to work particularly well when the clause set is (Horn or) near-Horn, in a suitable sense. To emphasize that this system is intended for clause sets that are not far from Horn sets we call the system *near-Horn Prolog* (or *nH-Prolog*). A modest extension of what we currently view as the most attractive implementation version of nH-Prolog is first-order complete (as standard theorem provers are) if iterative deepening is used as a search control rather than the depth-first search employed by Prolog. (The system is complete as a refutation system, but not capable of finding all logical consequences.)

Prolog has proved to be amazingly successful considering that it is the first implementation of the concept of logic programming. In spite of obvious flaws in the nature of depth-first search, unsound unification (this flaw has to go!), and the restriction to Horn clauses, the success of Prolog has been dramatic. There are good reasons for this. First is the surprising robustness of Horn clause programs for representing what we wish to compute; second, and closely related, is the simplicity of the depth-first problem reduction format such a restriction allows. Third, and closely related also, is the great speed of processing that can be realized by clever implementation techniques exploiting the inherent simplicity. Extensions to Prolog to date have not been readily accepted because they violate too strongly the properties that make Prolog so successful. The device "negation as failure" [1] survives because, in spite of obvious flaws, pragmatically it works better than its alternatives. We will propose a honest negation that appears to work as computationally effectively as negation as failure. (Negation as failure does retain a role here, for use with negative queries.)

Prolog-like systems that are complete proof procedures are well-known. Indeed, the group that designed PROLOG (Colmerauer, Roussel, etc.) first modified an SL-resolution theorem prover [6] built by Roussel [5]. Stickel's Prolog Technology Theorem Prover (PTTP) directly extends pure Prolog to a complete system (using an iterative deepening or recomputed depth-first search rather than straight depth-first search). The two systems mentioned above are closely related in concept because SL-resolution is a modification of the Model Elimination (ME) procedure [7], [8] which is the basis of the PTTP system. While the ME extension to linear input resolution (the heart of PROLOG) has real attractions, it is slower than desirable due to an auxiliary list, often of substantial size, that must be scanned at each call. We clarify this later. We also have an auxiliary list but the need to access it is quite limited.

2. SOME APPROACHES TO EXTENDING PROLOG.

In this section we review the two methods of extending Prolog mentioned earlier, negation as failure and ME (PTTP), as important extension methods our new method must be measured against. (There are other methods of extension suggested recently, for example, see [3], [4]. A new general theorem-proving method that extends Prolog along the lines of [10] has been announced in Plaisted [11].) The method of presentation will be a running example, admittedly a trivial one, both for clarity and because of space limitations.

The notation we use is the Edinburgh version, close to Clocksin and Mellish [2]. We assume familiarity with this terminology for Prolog. The example is that used in Loveland [8] to illustrate how ME extends the problem reduction method. The example is propositional, but a later example illustrates some characteristics of our system for non-ground presentations.

Figure 1 presents our running example, where the usual propositional letters denoting statements are replaced by Prolog atoms. The axioms are presented in typical propositional form; the traditional associated propositional assertion for the problem can be formed as an implication, with the conjunction of axioms as antecedent and the atom "wet" as conclusion.

Axioms

1. I have a swimming pool.
2. If I have a swimming pool,
   and it doesn't rain,
   then I will go swimming.
3. If I go swimming,
   then I will get wet.
4. If it rains,
   then I will get wet.

Query. Do I get wet?

s_p:	I have a swimming pool.
sw:	I go swimming
wet:	I get wet
rain:	It rains

Propositional presentation

Axioms

1. s_p
2. s_p & ¬rain ⊃ sw
3. sw ⊃ wet
4. rain ⊃ wet

Query: wet?

A non-Horn problem

Figure 1

Figure 2 presents a demonstration that the query succeeds in our example problem using the negation as failure protocol. The input is the program and query, using the *not* operator employed by the negation as failure method. The deduction notation is a straightforward extension of the Clocksin-Mellish notation (without periods, as no end-of-line markers are needed). Step 4 is the subdemonstration attempt to prove the goal *rain* ; the failure to prove this goal establishes the complement goal *not (rain)* and the original query succeeds. Notice that the reasoning is invalid; axiom iv is not used here although it is needed for a valid conclusion! It is relevant to later comparison that each *not* instantiation sets up a new proof search, indeed one that must exhaust the search space for the intended negated conclusion to be reached.

Program:

i) s_p.
ii) sw :- s_p, not (rain).
iii) wet :- sw.
iv) wet :- rain.

Query: ?-wet.

Deduction:

0)	?- wet	
1)	:- sw	by iii
2)	:- s_p, not (rain)	by ii
3)	:- not (rain)	by i
	———	
4)	?- rain fail	initiate new deduction
	———	
5)	Yes	step 4) fails

Negation as failure

Figure 2

Figure 3 presents the ME approach to handling non-Horn clause sets. Clause ii) now has the formal negation symbol rather than an operator for *not*, and the procedure incorporates the standard semantics of negation of propositional and predicate logic. Contrapositives are needed at least in sufficient number to insure that every negated goal also appear as the head of some clause. (This is not quite strong enough.)

Program

i) s_p.
ii) sw :- s_p, ¬rain.
iii) wet :- sw.
iv) wet :- rain
iv') ¬rain :- ¬wet contrapositive of iv

Query:

?- wet.

Deduction:

Aux List			
	0)	?- wet.	1
wet	1)	:- sw	by iii
sw,wet	2)	:- s_p, ¬rain	by ii
sw,wet	3)	:- ¬rain	by i
¬rain,sw,wet	4)	:- ¬wet	by iv'
	5)	Yes	mate in list

Model Elimination (PTTP) approach

Figure 3

The deduction mechanism uses an auxiliary list wherein the calling goal is added when the body of the called clause replaces the calling goal in the next statement in the deduction. If any goal of the called clause is complementary (a *mate)* to a member on the auxiliary list, then the goal is not added to the new statement. (We represent it as added and then deleted, for clarity of exposition). We note that the first-order case requires unification of incoming goals with auxiliary list members to be considered so branching of the process is necessary if a non-null substitution is used. This is clearly necessary because the mating may be incorrect and the instantiations in error. (If the mating involves no instantiation then deletion is always correct, so no branching is necessary.)

The advantage of this procedure is clear; it is a direct extension of Prolog with a correct semantics for negation. The disadvantages are the need for contrapositives in the input set (which may include the negated goal on occasion) and the need to process the auxiliary list with each call. Clever programming tricks (e.g., hashing predicate names) can greatly reduce the average time spent processing the list but the list length grows with the depth of each branch of the search tree, which can be extensive for many (most?) large programs.

Although the ME approach avoids the problem of the *closed world assumption* inherent in negation by failure, the cost in computation time is apparently too great to make the ME approach attractive in typical database settings. In particular, the full auxiliary list appears to be needed even if the program is almost Horn, e.g. only one or two negations might appear in the input clause set (before contrapositives are added). The system we introduce in the next section has its computational effort grow no faster (and often more slowly) than the number of negations that appear when prepared for the ME process.

3. NAIVE nH-PROLOG.

The first distinguishing characteristic of the nH-Prolog methods is that no negation symbol, contradiction symbol or negation operator need be formally introduced. (One device used is close to a contradiction symbol in spirit, however.) The nH-Prolog system is a positive implication logic (all heads and goals are positive literals) but negated facts and similar constructs are appropriately handled by a simple encoding. Indeed, any first-order formula in conjunctive normal form to be tested for unsatisfiability is easily converted to a program with a (dummy) query which succeeds if and only if (iff) the clause set is refutable. In this section we present a variant of nH-Prolog that lacks a refinement used in more effective nH-Prologs, so we call this variant *naïve nH-Prolog.*

The sole new architectural feature to the input set (program) relative to Prolog is the use of multiple heads instead of a single head. If a clause has multiple heads they are separated by a semi-colon, the Edinburgh (Clocksin-Mellish) symbol for "or", which is the correct semantics.

Figure 4 presents our example in the format for nH-Prolog. The second clause is now a modification of the sentence it captures, i.e. "If I have a swimming pool and it doesn't rain then I go swimming". The modification, made because negation is not directly expressible, is to encode the equivalent sentence "if I have a swimming pool then either it rains or I go swimming". It is a well-known valid transformation to change literals to the opposite side of an implication while deleting or adding a negation sign.

Program:

i) s_p.
ii) sw; rain :- s_p.
iii) wet :- sw.
iv) wet :- rain.

Query: ?- wet.

Deduction:

0)	?- wet	
1)	:- sw	by iii
2)	:- s_p # rain	by ii
3)	:- # rain	by i
4)	:- wet # rain	restart
5)	:- rain # rain	by iv
6)	Yes	cancel

nH-Prolog

Figure 4

We now discuss the manner of deduction for *naïve nH-Prolog* as illustrated here. As usual, one begins with the query, which calls a clause in usual Prolog fashion (line 1). Goal *sw* calls clause ii which has two heads. In clause ii *sw* is the *called head* and *rain* is the *auxiliary head.* (In general there may be more than one auxiliary head in a clause.) The body of the clause replaces the goal, in normal Prolog fashion. The auxiliary head is written to the right of the # sign, called the *wall,* and is referred to as the *deferred head.* The leftmost goal then calls a

clause to continue the deduction. When a line is reached with no goals and no deferred heads, the deduction is successfully completed. If the line has only deferred heads, then a *block* is completed and a new block begins by reentering the query as a single goal, while retaining all deferred heads from the previous line. All blocks but the first are called *restart blocks* because one restarts with the query. (The first block is the *start* block.) Thus line 4 is labeled "restart". Goal *wet* can call clause iv with goal *rain* as its body. Whenever a goal agrees with (is identical to) the leftmost deferred head both the goal and the deferred head are removed by an action called *cancellation*. (In the first-order case this would involve unification and, if a proper substitution is made, branching of the process will occur.) In this example this leaves no goal or deferred head so the query succeeds.

The device of recording deferred heads and deferring attention to them until all goals and their descendents are removed, together with the cancellation method for elimination of deferred heads, constitutes the mechanism of nH-Prolog. There are conditions and concerns which we address below, and an added mechanism is used when we leave naïve nH-Prolog. At this time we point out the conceptual simplicity of this addition and emphasize the advantage of deferring action on the non-Horn part until the Horn part is completed. Normal backtracking will occur from many false paths by the standard Prolog mechanism and the deferred heads are pursued only when the goal set is empty. There is an added literal to check at each call, the leftmost deferred head. (However, not all deviations from a Horn clause set introduce auxiliary heads, as we see later.)

We present here the form of a successful deduction, not methods for search for that deduction. This is customary in presenting logic systems. Many possibilities exist within this framework, not surprisingly. Since our primary motivation is to minimally extend Prolog the primary search mode we have in mind is depth-first search within each block, with usual backtracking upon failure. We call clauses in program order as Prolog does. We have indicated goal selection and deferred head selection order, which can be varied but is in keeping with Prolog conventions. (Canceling with an arbitrary deferred head brings up subtle issues not even hinted at here, but that variant has been considered.) Regular nH-Prolog initializes restart blocks differently; control strategies at those points are considered later.

To help the reader understand the declarative semantics of the nH-Prolog procedures we include a proof of soundness in this paper. Intuitively, the rough idea is simple. The start block represents a simple problem reduction proof that the query follows from the program as if the deferred heads did not exist. Each restart block ensures that the query still follows if the particular deferred head literal is true rather than the disjunctive partner used in the start block. Thus a form a splitting rule is being invoked. (However, the efficiency issue is more complex. It is like invoking the splitting rule only when one branch is trivialized. Amplification of the point is beyond our present space resources.)

Several conditions accompany the cancellation rule. Only (the descendents of) the leftmost deferred head at the restart line can be canceled within the block. We call the deferred heads of the restart time (and their descendents) *eligible* deferred heads although only the leftmost such actually may be used in canceling in the block. Newly created deferred heads of the block are ineligible for cancellation in the block and can be regarded as added at the end of the block, to the left of the existing deferred head list. The leftmost eligible deferred head can be used to cancel several goals, made identical by unification. (In hand proofs we have added new deferred heads to the deferred head list as created, and use some separating mark between the eligible and ineligible deferred heads.)

We also remark that no merging of goals or of deferred heads is needed, although merging can be done. If two deferred heads are identified the leftmost one must be the one retained, in part because one cannot break the "no cancellation of deferred heads in their introductory block" rule. The reason that it is desirable to not require merging is that at the general (first-order) level this is the factoring operation of resolution. That is, were merging required one would have to branch whenever a proper substitution would allow merging, because the alternate that the heads are not to be identified also must be considered.

We shift to a new, and trivial, example in Figure 5 to illustrate how negated facts are handled. The fact $\neg a$ is replaced by b :- a with the following justification: Given that $\neg a$ is true, then a implies anything so we will use $a \supset b$, written b :-a , where b is the query. That is, "the query if a " is implied by the fact $\neg a$. An indeterminant fact $\neg a \vee \neg c$ would be represented as b :-a ,c , where b is the query, because if $\neg a$ or $\neg c$ is true, then $a \& c \supset b$. Clearly, any disjunction of negative literals can be so represented. Thus we now know that every clause in a Skolem conjunctive form formula or clause set (i.e. the input to a resolution process; see [8]) can be transformed into the nH-Prolog format. Observe that every unsatisfiable clause set has at least one negative clause which, when transformed, has a (sometimes dummy) query as its head. Thus every unsatisfiable clause set translates into a sensible nH-Prolog program with query. As previously stated, there is an nH-Prolog deduction that will demonstrate that the query succeeds if the program comes from an unsatisfiable clause set. This is a completeness statement, and there is an issue regarding search control here. We postpone discussion of this point. We mention that there is a mechanism for overcoming the apparent disadvantage of naming the query in the program.

Axioms:

A $\vee$ B
$\neg$A

Query: B

Program:

i) a;b.
ii) b :- a. denotes $\neg$a

Query: ?-b .

Deduction:

0)	?-b	
1)	:- # a	by i
2)	:- b # a	restart
3)	:- a # a	by ii
4)	: Yes	cancel

Negated facts

Figure 5

We turn briefly to consider efficiency of computation relative to negation as failure. Negation as failure requires in effect a restart, attempting to prove the goal whose negation is asserted. A deferred head is in effect a negated goal and also involves a restart. Thus, on a rough scale there is comparable effort. However, nH-Prolog restarts involve comparisons against a deferred head list. On the other hand, negated facts do not generate deferred heads. Also, we suggest later an implementation format that avoids a full restart. Thus, which approach is more efficient awaits implementation experience.

We finally give a non-propositional example, in Figure 6. The example is very short but non-trivial in that an indefinite answer is expected. Notice that this is handled naturally in nH-Prolog by the two query entries. Each query requires a different substitution to execute the block in which the query is introduced. The disjunction of the query under all required substitutions is the appropriate answer. When one leaves the Horn clause domain one must be prepared for indeterminant answers.

Show:

P (a) \/ P (b) ⊃ ∃xP (x)

Program:

i) p (a);p (b).

Query:

? -p (X).

Deduction:

0)	?- p(X)	
1)	:- # p(b)	$X \leftarrow$ a
2)	:- p(Y) # p(b)	restart (new variables)
3)	:- Yes	$Y \leftarrow$b

X =a or X =b

Answer: p (a) \/ p (b)

Figure 6

It is instructive for the reader to compare this presentation of the problem of Figure 6 with the presentation of the same problem in [13], where the Prolog Technology Theorem Prover represents the way on ME system handles this problem. In this case the nH-Prolog system is somewhat more natural.

4. nH-PROLOG.

Whereas naïve nH-Prolog has pedagogical value it is both incomplete and relatively inefficient. Since the complete form of nH-Prolog is an extension of a more efficient Prolog, we want to implement a modification of the procedure of the previous section. We will specify nH-Prolog in a more formal manner and then outline the associated Soundness Theorem, which establishes that nH-Prolog only gives correct answers.

The primary modification is to possibly begin restart blocks with literals other than the query. These literals are, in a suitable sense, ancestors of the leftmost deferred head, which is the active deferred head. We define an *ancestor list* associated with each deferred head to be the ancestor list at the beginning of the block (which is empty for the start block) together with all the ancestor calling literals to the line that accepts the new deferred head.

Again, we present the procedure without control structure regarding choice of clause, to which we add a nondeterminism in choice of initial literal to begin a restart block. We do specify literal order within the *continuation* (the literals to the left of the #) and in the deferred head list within the line. Thus we present the correct conditions for a successful derivation, and leave search considerations to later.

Characteristics of an nH-Prolog deduction.

1. The first line contains only the query.
2. If line n contains a goal (i.e. the continuation is non-empty) then the leftmost goal is the calling goal. Line n+1 replaces the calling goal by the body of the called clause, otherwise inheriting the continuation of line n. As always, unification determines the proper instantiation, which is applied uniformly to all literals, in the continuation, the deferred head list and the ancestor list. The deferred head list is inherited from line n, with the possible addition of auxiliary heads from the called clause, the new deferred heads placed leftmost, to the right of the wall, but ineligible in this block. The new deferred heads are said to be *defined* at this line. The ancestor list for line n+1 is the ancestor list of line n plus the (instantiated) calling goal contained in line n. No merging of goals or deferred heads is required, but is optional; goals merge right, and deferred heads merge left.
3. A restart line follows a line with an empty continuation (no goals). The restart line has the deferred head list of the previous line. The ancestor list for this line is the ancestor list of the defining line of the leftmost deferred head of this restart line. The continuation is a literal chosen from the ancestor list for this line. (Note that the ancestor list may be a superset of the ancestor goals of the literal of the continuation.) However, the original query with new variables is used in place of the current instantiation of the original query when a query variant is required.
4. Each viable restart block contains one or more cancellations, all involving (instantiations of) the leftmost eligible deferred head of the current line and a literal of the body of the called clause used to create the line. The deferred heads canceled are always descendents of the same deferred head, the leftmost deferred head of the restart line. Thus each block yields a cancellation of precisely one deferred head, including various instantiations of that head. Unification may occur to allow cancellation. (Cancellation may be restricted to the leftmost literal of the continuation to speed processing.)
5. A line containing no deferred heads or goals is the concluding line of a successful deduction.

We now outline the soundness of the nH-Prolog system. A key notion in the proof is that of a query being falsifiable under a program. A query q is *falsifiable* under program P iff there is truth assignment to each literal such that q has (all instances with) valuation F and (every instance of) each clause of P has valuation T. We will show that an nH-Prolog deduction cannot terminate successfully if the query is falsifiable under the program.

Soundness Theorem. nH-Prolog is sound, i.e., if there is a successful nH-Prolog deduction of

query q from program P then an instance of q is logically implied by P.

Proof. First, we note that each deduction has a ground propositional deduction counterpart. Each time an instantiation occurs due to a non-null unification, the instantiation occurs throughout the deduction, including the deferred head lists, the ancestor tests, and all previous lines. (Of course, this need not be *explicitly* done in implementations.) Restart blocks start with the current instantiation. When the deduction concludes, assign an atomic constant to each variable uniformly.

Therefore, it suffices to argue soundness at the propositional level. We assume that we have a query q and ground program P for which a deduction has terminated successfully and that the q is falsifiable under P, to show this leads to a contradiction. We invoke the truth assignment that falsifies q; all references to values of literals now are to truth-values under this assignment.

We define a *critical head h* to be a deferred head with value T such that all literals in the ancestor list at the defining line for h have value F.

We prove the following statement by induction.

P(n): At line n either the continuation has (a literal with) value F or a critical head exists in the deferred head list.

Note that if P(n) holds for all n then the deduction cannot terminate successfully (as it never has an empty line), a contradiction.

Case 1. Start block. The continuation at line 1 has value F by definition because q has value F. Consider the clause q calls. Either there is an auxiliary head with value T (hence a critical head at line 2) or the disjunction of heads has value F so the clause body contains a literal with value F. Since a critical head, once created, cannot be removed in its block of introduction, we need only consider the alternative, that a false goal (goal with value F) exists at line 2. Choose the leftmost false goal g. All clauses until g becomes leftmost have a false continuation. When g calls a clause we have the same situation as when q was calling goal. Since now only q and g are ancestors, either a critical head is created or a new false literal is introduced into the continuation. This argument pertains to the remaining lines of the block. The block terminates with a critical head in the last line because the continuation is empty.

Case 2. The restart block does not have the critical head leftmost. Since only the leftmost deferred head is canceled, the critical head remains through the block.

Case 3. The restart block has the critical head leftmost. The ancestor list for the initial line of this block has all false literals because it is the ancestor list of the defining line for the critical head that is leftmost. Thus the initial continuation has value F. The argument follows that of the Start Block. One must notice that the cancellation does not endanger the development of the block because the critical head has value T so no false goal will be canceled. The all-false-literals of the initial ancestor list allows a critical head to be created following the argument of the Start Block. □

5. IMPLEMENTATION.

Although nH-Prolog is complete (see [9]) when a full unification is used and iterative deepening (or any equivalent search strategy) replaces depth-first search, we here are less

concerned with completeness than with minimally extending Prolog and with practical effectiveness, including speed of execution. For a description of a first implementation see [12]. The implementation, built directly on top of Prolog, uses depth-first search and has the option of full unification. Within a block the depth-first search is classical, identical to that of Prolog. The initialization of restart blocks involves some control decisions as to element of the ancestor list picked and the search starting point. We have adopted a less conventional strategy here, unusual enough to warrant labeling the variant; we call this version *progressive nH-Prolog*. We select as the initial goal the calling goal of the clause that defines the leftmost deferred head of the restart block, and, upon failures, work back up the ancestor list, deleting tried goals from the list. (This introduces a possible incompleteness; see [9]). Also, we begin the search for the next called clause not at the beginning of the clause list but at the clause following the defining clause for the leftmost deferred head (See [12]). Search stops at the last clause; currently, we do not wrap around to continue from the top. Although this introduces some extra incompleteness, there is evidence that the lost deductions are relatively few, due to multiple proof paths, and certainly the speed gain appears significant. (The wraparound is easily installed and is now an option.) Experimentation will confirm or refute its value of these devices.

Other implementation devices improve performance or convenience, such as the use of a system query to allow compilation of a program independent of the query (queries) asked. Clauses from negated facts can be built in. Space limitations forbid further discussion of these devices (see [9],[12]).

ACKNOWLEDGMENT

We thank Bruce T. Smith for several stimulating conversations regarding nH-Prolog, and for insights that led to improvements in Progressive nH-Prolog.

REFERENCES

[1] Clark, K.L. Negation as failure. *Logic and Databases* (eds. Gallaire and Minker), Plenum Press, New York, 1978, 293-321.

[2] Clocksin, W.F. and C.S. Mellish. *Programming in Prolog,* (2nd edition). Springer-Verlag, Berlin, 1984, xv + 297 pp.

[3] Gabbay, D.M. and U. Reyle. N-Prolog: an extension of Prolog with hypothetical implications, I. *J. Logic Programming 4,* 1984, 319-355.

[4] *IEEE Proceedings of the Symposium on Logic Programming,* Boston, Mass., July, 1985.

[5] Kowalski, R. Robert Kowalski on logic programming. *Future Generations Comp. Systems 1* (1), 1984, 79-83.

[6] Kowalski, R. and D. Kuehner. Linear resolution with selected function. *Artif. Intell. 2,* 1971, 227-260.

[7] Loveland, D.W. Mechanical theorem proving by model elimination. *J. ACM 15,* 1968, 236-251.

[8] Loveland, D.W. *Automated Theorem Proving: a Logical Basis.* North-Holland, Amsterdam, 1978, xiii + 405 pp.

[9] Loveland, D.W. Near-Horn Prolog and beyond. Forthcoming paper.

[10] Plaisted, D. A simplified problem reduction format. *Artif. Intell 18,* 1982, 227-261.

[11] Plaisted, D. Non-Horn clause logic programming without contrapositives. Preprint.

[12] Smith, B.T. and D.W. Loveland. An implementation of near-Horn Prolog. Forthcoming paper.

[13] Stickel, M.E. A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler. *Eighth Int'l Conf. on Auto. Deduction,* Lecture Notes in C.S. *230,* Springer-Verlag, Berlin, July, 1986, 573-587.

SECURITY CLASSIFICATION OF THIS PAGE

AD-A185 172

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	Unclassified
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
	ARO 21213.3-MA
6a. NAME OF PERFORMING ORGANIZATION: Duke University, Computer Science Dept. (6b. OFFICE SYMBOL (If applicable))	7a. NAME OF MONITORING ORGANIZATION: Army Research Office
6c. ADDRESS (City, State and ZIP Code): Durham, NC 27706	7b. ADDRESS (City, State and ZIP Code): P.O. Box 12211, Research Triangle Pk., NC 27709-2211
8a. NAME OF FUNDING/SPONSORING ORGANIZATION (8b. OFFICE SYMBOL (If applicable))	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: Grant DAAG29-84-K-0072
8c. ADDRESS (City, State and ZIP Code)	10 SOURCE OF FUNDING NOS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT NO
11. TITLE (Include Security Classification): Near-Horn Prolog	
12. PERSONAL AUTHOR(S): D.W. Loveland	
13a. TYPE OF REPORT: Technical / 13b. TIME COVERED: FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day): April 1987 / 15 PAGE COUNT: ii + 14
16. SUPPLEMENTARY NOTATION: Talk presented at the Fourth Int'l Conf. on Logic Programming, Melbourne, May 1987	

17 COSATI CODES		
FIELD	GROUP	SUB GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Prolog, Logic Programming, Intelligent data bases

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

We propose an extension to Prolog that handles clause sets that are Horn and almost-Horn in a manner that emphasizes clarity, efficiency, and minimal deviation from standard Prolog. Although one version of near-Horn Prolog is complete under appropriate search strategies, we focus on incomplete variants that allow fast processing. Negation in the program is handled correctly, although within a positive implication logic extension of Horn logic. Processing speed degradation is directly proportional to the distance the program is from a Horn clause set.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS ☐	21 ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL: Donald W. Loveland	22b TELEPHONE NUMBER (Include Area Code): 919-684-3048 / 22c OFFICE SYMBOL

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE

END

11-87

DTIC